Towards a versatile exascale operating system to support on-line load and failure management

Hermann Härtig*, Amnon Barak**, Wolfgang E. Nagel*, Alexander Reinefeld***
* TU Dresden ** Hebrew University Jerusalem *** Konrad Zuse Institute Berlin
Contact: hermann.haertig@tu-dresden.de

We believe that increasing failure rates and the increasing relevance of load imbalance and noise will be major obstacles for exascale computing. We also believe that – even if novel programming paradigms will be found – HPC programs with tight computation/communications cycles, for example represented by applications based on an MPI-library, will be among the most demanding applications. We are convinced that static partitioning together with application-level load balancing as used in current high-end machines will cease to be effective.
In contrast, we envision future exascale computers, where applications of fairly different types will share an exascale computer. They will flow in and out, expand and shrink on demand. These applications will run on dedicated application-specific run-times, some of them extremely small and optimized for an application domain, others containing a full components-off-the-shelf operating system (COTS) such as GNU/Linux. We assume these run-times to be based on a thin common software substrate and will range from library-style implementation techniques to (thin) virtualization architectures. We consider it a key challenge to invent interfaces and protocols that allow applications and/or run-time to cooperate with the thin common software substrate in managing load and failure handling.
The hypothesis underlying our approach is that the basic technologies needed for such an exascale operating system are available and mature, but need to be adapted and integrated. These technologies are microkernels and microkernel-based virtualization(such as L4 and L4Linux[1,7]), erasure-code based storage (such as used in RAID), on-line management systems (such as load balancing in MosiX[2]), versatile distributed storage systems (such as XtreemFS[3,4]), split applications (as found in Real-Time and High-Security architectures, for example based on L4[5,6]) and application-specific libraries (such as OpenMPI).

The technology combination is straight forward (see figure at the end of page 2 for a rough sketch):

- All nodes of an exascale computer share a common software substrate consisting of a microkernel and few small and predictable dedicated components.
- Cores can be computation or service cores. Service cores run a (potentially fat) COTS operating system for overall management. Computation cores run (hopefully slim) components, just the minimum needed for the performance critical parts of an (HPC) application.
- Demanding applications (or their libraries) are split into critical and uncritical parts. Uncritical parts run on service nodes and COTS operating systems (for example Linux), critical parts on a dedicated operating system personality (for example: library-based). This approach has been successfully demonstrated in other areas[5]. Obvious choices for splitting are a split implementation of an MPI library and of a storage system used to handle redundant cross-node checkpoints. If more computing power is needed for applications needing COTS operating systems, thin virtual machine instances (for example as built for the NOVA virtualization system[7]) are created or migrated to the computation nodes.
- To cope with node failures, checkpoints are written distributed onto nodes (we expect to have massive low-power storage on each node, for example low power RAM or PCM). To protect checkpoints from node failures, erasure codes are added (just like in RAID) and striped on different nodes. Applications need not to write checkpoints to file systems, but should be enhanced such that can produce a dense checkpoint upon request by management software.
- Applications and on-line systems management (for example with scalable gossip-based distributed bulletin board and heuristics such as in Mosix) cooperate. For example, applications "yell for help" if computation demand explodes, then system-level management distributes load on available nodes.

None of the technologies is new, but – to our surprise – they have never been properly integrated. The resulting system – rather than claiming to solve all exascale problems - forms a substrate that can be used to explore many additional opportunities. These include but are not limited to incremental checkpoints, transparent active replication (as has been demonstrated for embedded systems on top of L4[8]), plugin modules for other forms of load management, split libraries for other programming paradigms.

Related work:

Some related work has already been mentioned above and will be mentioned in the assessment section. Obvious other related work includes microkernels used in HPC. Blue Gene's CNK[9] is an example for an extremely thin kernel, however providing no protection which – in our view - will be needed. First-generation kernels, for example such based on Chorus[10], are much larger and less efficient than second and third generation kernels such as those of the L4 family[11]. Commercial virtualization systems are an order of magnitude larger than experimental novel ones [7,12]. Some experts recognized the need to relieve applications programmers from balancing issues by factoring it out into libraries(Charm ++,[13]). However, we believe, dedicated interaction is needed between an operating system directly controlling hardware and the applications to enable informed decisions on failure and load handling.

Assessment:

Challenges addressed: Load imbalances are expected from highly dynamic applications such as those based on dynamic multi grid algorithms. An application-level management alone will not work any more in exascale. Writing checkpoints to file systems will be no longer effective as higher failures rates together with larger applications will require frequent checkpoints (or an alternative method such as active replication).
Mature technologies are combined in this approach. MosiX is used extensively in large scale applications. Implementations of the L4 family of microkernels run in many research labs, in over a billion cell phones, have been used in HPC experiments (for example in IBM's Kitty Hawk project on Blue Gene). Erasure-code based storage is widely used in RAID technology. XTreem FS is used as basis for many Cloud applications. Open-MPI and other MPI libraries form the basis for many HPC applications.
Neither uniqueness nor novelty can be claimed for any of the used technologies. A competent combination however has never been attempted to the best of our knowledge.
We expect our approach to be applicable wherever multiple dynamic applications share large systems whose components can fail.
Efforts – based on initial analysis – could be confined to a small team of 5 expert engineers to build a quick and dirty, limited prototype of an integrated system and produce a detailed design document within a year. A larger team of approximately 10 can produce a usable prototype in subsequent 2 years. Main risk for this effort estimate is the availability of precise documentation of hardware, support by equipment vendors and sufficient time on a large HPC. The authors currently seek such funding.
Other challenges, for example the usage of special purpose hardware devices (GPU, flexible hardware) are largely orthogonal to our approach.

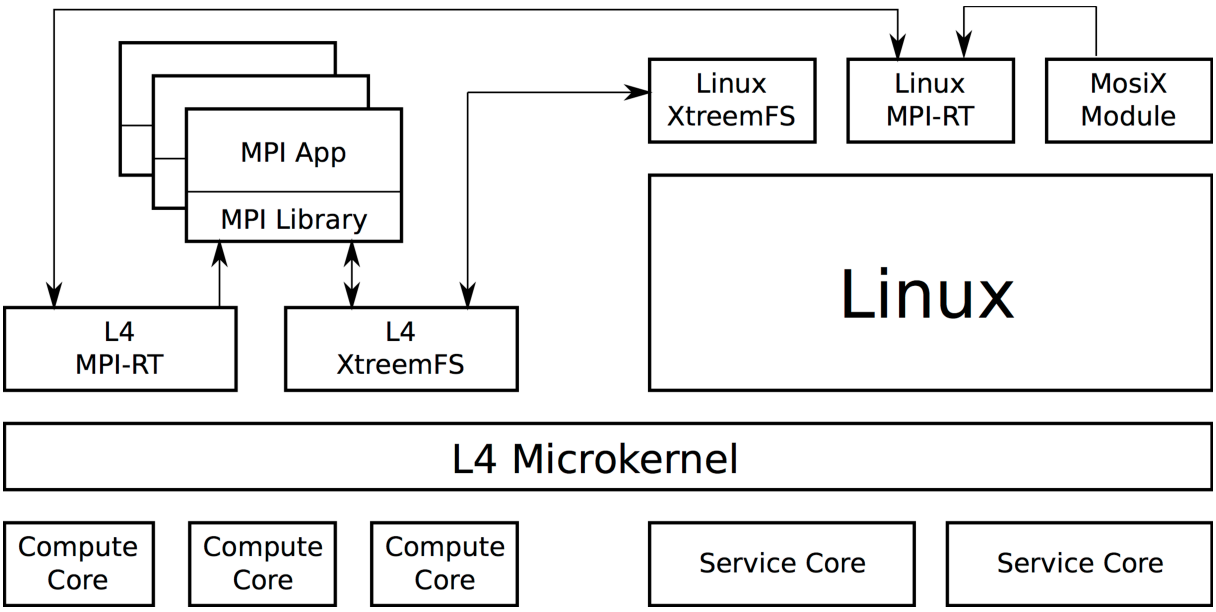

Figure: Architecture Overview

References:

[1] H. Härtig, M. Hohmuth, J. Liedtke, S. Schönberg, and J. Wolter. The performance of µ-kernel-based systems. In Proceedings of the 16th ACM Symposium on Operating System Principles (SOSP), pages 66–77, Saint-Malo, France, October 1997.

[2]MosiX Cluster Operating System. http://www.MOSIX.cs.huji.ac.il. last checked: 05/15/2012.

[3] XtreemFS - A Cloud File System, http://www.xtreemfs.org, last checked: 05/15/2012.

[4] K. Peter and A. Reinefeld. Consistency and fault tolerance for erasure-coded distributed storage systems. In 5th Intl. Workshop on Data Intensive Distributed Computing (DIDC 2012) at HPDC 2012, page to appear, 2012.

[5] Lenin Singaravelu, Calton Pu, Hermann Härtig, and Christian Helmuth. Reducing tcb complexity for security-sensitive applications: three case studies. In Proceedings of the 1st ACM SIGOPS/EuroSys European Conference on Computer Systems 2006, EuroSys '06, pages 161–174, New York, NY, USA, 2006. ACM.

[6] Carsten Weinhold and Hermann Härtig. VPFS: Building a Virtual Private File System with a Small Trusted Computing Base. In Proceedings of the 3rd ACM SIGOPS/EuroSys European Conference on Computer Systems 2008, Eurosys '08, pages 81–93, New York, NY, USA, 2008. ACM.

[7] Udo Steinberg and Bernhard Kauer. NOVA: a microhypervisor-based secure virtualization architecture. In EuroSys '10: Proceedings of the 5th European conference on Computer systems, pages 209–222, New York, NY, USA, 2010. ACM.

[8] Björn Döbel, Hermann Härtig, and Michael Engel. Operating system support for redundant multithreading. In Proceedings of the 12th Conference on Embedded Software, EMSOFT, New York, NY, USA, 2012. ACM. To appear.

[9] George Almasi, Ralph Bellofatto, Jose Brunheroto, Calin Cascaval,Jose G. Castanos, José G, LuisCeze, Paul Crumley, C. Christopher Erway, Joseph Gagliano, Derek Lieber, Xavier Martorell, José E. Moreira, and Alda Sanomiya. An overview of the blue gene/l system software organization. In In Proceedings of Euro-Par 2003 Conference, Lecture Notes in Computer Science, pages 543–555. Springer-Verlag, 2003.

[10] M. Rozier, V. Abrossimov, F. Armand, I. Boule, M. Gien, M. Guillemont, F. Herrmann, C. Kaiser, S. Langlois, P. Léonard, and W. Neuhauser. Overview of the chorus distributed operating systems. Computing Systems, 1:39–69, 1991.

[11] J. Liedtke. On µ-kernel construction. In Proceedings of the 15th ACM Symposium on Operating System Principles (SOSP), pages 237–250, Copper Mountain Resort, CO, December 1995.

[12] VMware, http://www.vmware.com, last checked 07/10/2012

[13] Gengbin Zheng, Lixia Shi, and Laxmikant V. Kalé. FTC-Charm++: An In-Memory Checkpoint-Based Fault Tolerant Runtime for Charm++ and MPI. In 2004 IEEE International Conference on Cluster Computing, pages 93–103, San Diego, CA, September 2004.